# Escapy2.0 Engine
# Short Specification and User Guide

Генрих Тимур Домагальски

31.07.2017 Издаине 1

# Содержание

# О движке

Escapy2 это игровой движок написанный на java с использованием библиотек Dagger1, libGDX и Gson. Поскольку libGDX является лишь низкоуровневой оберткой над lwjgl - движок дает полноту простора в использовании openGL, в свою очередь Dagger делает код более модульным и масштабируемым. На момент издания этого документа, движок состоит из пяти ключевых пакетов:

1. Context
2. Desktop
3. Graphic
4. Group
5. Utils

Каждому из вышеперечисленных пакетов посвящена отдельная глава, подробнее со структурой api можно ознакомится через javadoc. На конец следует сразу заметить, что данный документ, так же как и сам движок расчитан на разработчиков неплохо знакомых с java и ООП, а так же основами openGL. Основной задачей документа не является скурпулезное описание API - за этим следует идти в javadoc, основная же цель документа - в первую очередь кратко обрисовать возможности движка, его принцип работы, а так же life-cycle и тп.

# 1 | Начало работы

Вход производится похожим образом как в libGDX и lwjgl - в main с созданием инстанции **LwjglApplication**. Для этого создается объект **LwjglApplicationConfiguration** который загружается из json файла с помощью **EscapyDesktopConfigLoader**, о самих загрузчиках и механизме сериализации в движке более подробно потом.

```
import ...

public class DesktopLauncher {

    public static void main (String[] arg) throws Exception {

        LwjglApplicationConfiguration config = DesktopConfigLoaderBuilder.Default()
                .setLoadedClass(LwjglApplicationConfiguration.class)
                .setSerializedClass(SerializedDesktopConfig.class)
                .setPath(System.getProperty("user.dir") + separator + "Configuration.json")
                .setName("MainConfiguration")
                .build()
        .loadDesktopConfig();

        new LwjglApplication(new EscapyApplicationAdapter(MainEnvironment.class, new MainModule()), config);
    }
}
```

```
{
    "type": "EscapyConfiguration",
    "name": "MainConfiguration",

    "resizable": false,
    "vSync": true,
    "fullscreen": false,
    "forceExit": true,
    "useGL30": true,

    "scrWidth": 1920,
    "scrHeight": 1080,

    "fps": 25
}
```

При создании **LwjglApplication** в качестве аргумента передается **EscapyApplicationAdapter**, который в свою очередь в качестве аргумента принимает класс наследующийся от **EscapyGameContext** и varargs модулей Dagger'a.

```
import ...

public class MainEnvironment extends EscapyGameContext {

    private final EscapyScreen initialScreen;

    @Inject
    protected MainEnvironment(
            Collection<EscapyScreen> escapyScreens,
            EscapyScreen initialScreen,
            EscapyGameContextConfiguration contextConfiguration) {

        super(escapyScreens, contextConfiguration);
        this.initialScreen = initialScreen;
    }

    @Override
    protected EscapyScreen getInitialScreen() { return initialScreen; }

}
```

Подробнее о том как использовать модули Dagger'a можно прочитать на оффициальном сайте проекта (*http://square.github.io/dagger/*). **EscapyGameContext** имеет два конструктора, один из них как аргумент принимает инстанцию класса унаследованного от **EscapyGameContextConfiguration** - абстрактного класса предоставляющего конфигурацию проекта через методы которые можно перегрузить в случае необходимости.

```
import ...

public class MainConfiguration extends EscapyGameContextConfiguration {

    @Override
    public String getResourcesDir() {
        String local = System.getProperty("user.dir").replace( target: separator+"core"+separator+"assets",  replacement: "");
        return local + separator + "res";
    }

    @Override
    public String getConfigsFilePath() { return this.getResourcesDir() + separator + "configurations"; }

    @Override
    public void configurePropertyKeys(PropertyKeysStorage propertyKeyStorage) {
        propertyKeyStorage
                .addPropertyKey("BLEND_SHADERS_ROOT_DIR_PATH")
                .addPropertyValue(getResourcesDir() + separator + "shaders" + separator + "blend")
        .save();
    }
}
```

# 2 | Context

Самый главный и значимый пакет движка в плане его архитектуры. Его основными элементами являются два субпакета - ***game*** и ***annotation*** и класс ***EscapyGameContext***. Последний наследуется от интерфейса ***EscapyScreenContext*** позволяя тем самым на работу с экранами (сценами).

## 2.1 Game

Основные эллементы данного субпакета это классы:

- EscapyGameContextConfiguration - абстрактный класс делегирующий настройки
- EscapyScreenContext - интерфейс управления экранами
- EscapyScreen - интерфейс экрана (сцены).
- PropertyKeysStorage - интерфейс позволяет сохранять пары ключ-объект.
- Escapy - синглетон хранящий некоторые настройки.

### EscapyScreen

Отдельного внимания заслуживает этот интерфейс, он в свою очередь наследуется от интерфейса ***Screen*** из библиотеки libGDX и содержит callback методы в которых должна находиться логика игры. Класс реализующий данный интерфейс, может (опционально) быть отмечен аннотацией ***@SreenName("...")***, в таком случае этому экрану будет присвоенно имя с помощью которого к этому экрану можно будет обращаться через методы интерфейса ***EscapyScreenContext***.

## 2.2 Annotation

Содержит аннотации такие как ***@SreenName("...")***, а так же субпакет ***meta*** содержащий процессор аннотаций построенный по шаблону «Декоратор». Если интересуют подробности или возникло желание написать свою собственную имплементацию, то лучшем решением будет отсылка в javadoc или исходники.

# 3 | Utils

Пакет со вспомогательными классами и прочими полезными вещами. Особого внимания заслуживают:

- ***EscapyArray* и *EscapyAssociatedArray*** - интерфейсы (и их реализации) наследующие Iterable с массивом внутри.
- Пакет ***proxy*** - позволяет инстацировать объекты с listener'ами внутри.
- ***EscapyInstanceLoader*** - позволяет инстанцировать объекты по имени с помощью аннотации ***@EscapyInstanced("...")*** или по имени метода.
- ***EscapySerialized* и *EscapySimpleSerialized*** - интерфейс и абстрактный класс реализующий этот интерфейс, служат шаблоном для сериализуемы с помщью Gson'а классов.

## 3.1 EscapySimpleSerialized

Так выглядит этот шаблон в исходниках.

```java
import ...

public abstract class EscapySimpleSerialized implements EscapySerialized {

    @SerializedName("type") @Expose public String type = "";
    @SerializedName("name") @Expose public String name = "";
    @SerializedName("attributes") @Expose public List<String> attributes = new LinkedList<>();

    @Override public Collection<String> getAttributes() {
        return attributes;
    }

    @Override public String getName() {
        return name;
    }

    @Override public String getType() {
        return type;
    }

}
```

А так выглядит его json.

```json
{
    "name": "",
    "type": "",
    "attributes": ["", "", ""]
}
```

Поскольку все классы движка должны сериализовываться через этот шаблон, код выше является необходимым минумом, для того что бы загрузчики движка могли успешно выполнить свою работу.

## 3.2 EscapyInstanceLoader

Класс реализующий этот интерфейс позволяет на вызов инстанцирующих методов по имени либо самого метода, либо указанного в аннотации которым этот метод отмечен. Этот механизм очень удобен в использовании в загрузчиках движка и потому повсеместно там используется - для инстанцирования объектов по имени указаноому в json файле, либо для загрузки атрибутов для уже существующего объекта.

### 3.2.1 Загрузка аттрибутов

Данный пример показывает как производить загрузку аттрибутов для уже существующего объекта - сначала создается класс реализующий интерфейс, затем инстанция класса передается в загрузчкик.

```
import ...

public class SimpleShiftLogic implements EscapyInstanceLoader<LayerShiftLogic> {

    private final EscapyCamera camera;
    public SimpleShiftLogic(EscapyCamera camera) { this.camera = camera; }

    @EscapyInstanced("move")
    public LayerShiftLogic moveShiftLogic(LayerShiftLogic shiftLogic) {

        return new LayerShiftLogic() {
            float[] initial = camera.getPosition();

            @Override
            public float[] calculateShift(LayerShift shift) {
                float[] position = camera.getPosition();
                float tx = position[0] - initial[0];
                float ty = position[1] - initial[1];
                float[] direct = shift.getDirect();
                return new float[]{tx * direct[0], ty * direct[1]};
            }
        };

    }
}
```

И в загрузчкие, во время инициализации используется для создания нужного объекта по его имени взятом из json файла, посредством вызова метода:
***T: objectToLoad = loadInstanceAttributes(T: objectToLoad, String[]: attributes);***

```
        LayerShiftLogic shiftLogic = shift -> new float[]{0,0};
        if (shiftLogicAttributeLoader != null)
            shiftLogic = shiftLogicAttributeLoader.loadInstanceAttributes(shiftLogic, serializedShift.attributes);
        shifter.setLayerShiftLogic(shiftLogic);
```

### 3.2.2 Интсанцирование

Инастанцирование производится по потому же принципу что и загрузка аттрибутов, с той лишь разницей, что метод отмеченный аннотацией ***@EscapyInstanced("...")*** не имеет аргументов.

```java
import ...

public class ExampleClassInstancer implements EscapyInstanceLoader<SomeExampleClass> {

    @EscapyInstanced("example_one")
    public SomeExampleClass randomMethodNameOne() {
        return new SomeExampleClass( name: "Say: 1");
    }

    @EscapyInstanced("example_two")
    public SomeExampleClass randomMethodNameTwo() {
        return new SomeExampleClass( name: "Say: 2");
    }
}
```

В свою очередь создание инстанции нужного нам объекта происходит через вызов метода:
***loadInstance(String: instanceName, Object[]: args);***
или просто:
***loadInstance(String: instanceName);***

```java
import ...

public class TestClassOne {

    @Test
    public void test() {

        EscapyInstanceLoader<SomeExampleClass> instancer = new ExampleClassInstancer();

        SomeExampleClass exampleClassOne = instancer.loadInstance( name: "example_one");
        SomeExampleClass exampleClassTwo = instancer.loadInstance( name: "example_two");

        System.out.println(exampleClassOne); <- Say: 1
        System.out.println(exampleClassTwo); <- Say: 2
    }

}
```

# 4 | Desktop

На данный момент, этот пакет служит только для загрузки начальной конфигурации из json файла в desktop-версии приложений. Для этого используются такие интерфейсы (их реализации) как ***DesktopConfigLoader и DesktopConfigLoaderBuilder***. Очевидный пример использования этого пакета продемонстрирован в самом начале документа.

```
import ...

public class DesktopLauncher {

    public static void main (String[] arg) throws Exception {

        LwjglApplicationConfiguration config = DesktopConfigLoaderBuilder.Default()
                .setLoadedClass(LwjglApplicationConfiguration.class)
                .setSerializedClass(SerializedDesktopConfig.class)
                .setPath(System.getProperty("user.dir") + separator + "Configuration.json")
                .setName("MainConfiguration")
                .build()
        .loadDesktopConfig();

        new LwjglApplication(new EscapyApplicationAdapter(MainEnvironment.class, new MainModule()), config);
    }
}
```

Builder создает нужную нам инстанцию загрузчика, после чего остается вызвать на этой инстанции метод «***loadDesktopConfig();***» который заинстанцирует нужный нам объект и устновит в нем значения полей считанные из json файла.

# 5 | Graphic

Пакет Graphic состоит из трех субпакетов:

- ***Camera***
- ***Render***
- ***Screen***

## 5.1 EscapyCamera

Класс из пакета ***Camera*** инкапсулирующий ***OrtographicCamera*** с дополнительным функционалом - простота и удобство, рекомендуется к использованию.

## 5.2 Render

Ключевой субпакет, на данный момент состоит из субпакетов:

- ***Fbo***
- ***Light***
- ***Mapping***
- ***Program***

### 5.2.1 Mapping

Данный пакет включает в себя 4 интерфейса, три из них содержат методы вызываемые во время отрисовки:

- ***GraphicRenderer*** - методы этого интерфейса вызываются во время отрисовки цветных(обычных) текстур объектов.
- ***NormalMapRenderer*** - методы вызываются во время отрисовки текстур карты нормалей.
- ***LightMapRenderer*** - методы вызываются во время отрисовки текстур карты света.
- ***EscapyRenderable*** - этот интерфейс наследуется от трех выше перечисленных.

### 5.2.2 FBO

FBO - иначе Frame Buffer Object, в движке представлен интерфейсом ***EscapyFBO*** и его стандартной реализацией ***EscapyFrameBuffer*** которая инкапсулирует ***FrameBuffer*** из библиотеки libGDX, но с дополнительным полезным и удобным функционалом. О том как работают FrameBuffer'ы следует ознакомится самостоятельно через материалы посвященные ***openGL***.

### 5.2.3 Program

Состоит из двух субпакетов ***gl10*** и ***gl20***. Первый использует нативные вызовы openGL без шейдеров в процессе рендеринга, второй в свою очередь нацелен на использование именно шейдеров.

#### GL10

Функционалом gl10 пользуются такие классы как например:

- ***EscapyGLBlendRenderer*** - интерфейс ответственный за блендинг.
- ***NativeSeparateBlendRenderer*** - нативная реализация интерфейса выше
- ***LightMask*** - маска, используется для затемнения активной области экрана.

#### GL20

Пакет направленный на работу с шейдерами удобным способом. Работа осуществляется посредством двух основных интерфейсов ***EscapyShader*** и ***UniformsProvider***, а так же интерфейсов от них налседующихся как ***EscapySingleSourceShader*** и ***EscapyMultiSourceShader***. Работа с юниформами (их загрузка и тп) осуществляется посредством вспомогательного класса ***StandardUniforms*** и ***Uniform<T>*** внутри него.

```
private void initBlender(ShaderFile shaderFile) {

    StandardUniforms uniforms = uniformBlender.getStandardUniforms();
    uniforms.addFloatUniform( name: "u_coeff");
    uniforms.addFloatUniform( name: "u_angCorrect");
    uniforms.addFloatArrayUniform( name: "u_color");
    uniforms.addFloatArrayUniform( name: "u_fieldSize");
    uniforms.addFloatArrayUniform( name: "u_umbra");
    uniforms.addFloatArrayUniform( name: "u_radius");
    uniforms.addFloatArrayUniform( name: "u_angles");

    uniformBlender.setSourcesNames("targetMap", "u_lightMap");
    uniformBlender.loadProgram(shaderFile);
}
```

Выше изображен пример использования класса ***StandardUniforms***.

В целом для работы с шейдерами достаточно двух стандартных реализаций интерфейсов ***EscapyUniformSingle*** и ***EscapyUniformBlender***. Их реализации предоставленные движком это ***SingleRendererExtended*** и ***BlendRendererExtended*** соотвественно. Достаточно в аргументах конструктора этих классов указать файлы .vert и .frag шейдеров, а с помощью метода ***getStandardUniforms()*** установить значения для юинформов. Хорошим примером может послужить исходный код класса ***EscapyLightSource***.

### 5.2.4 Light

Данный пакет как можно догадаться из названия служит работе со светом.
Субпакет ***source*** отвечает за создание источников света с помощью классов ***EscapyLightSource*** и ***LightSource*** (рекомендуется использовать второй). Субпакет ***processor*** оветчает за правильную отрисовку источников света посредством интерфейса ***EscapyLightProcessor*** и его двух стандартных реализаций ***EscapyFlatLight*** и ***EscapyVolumeLight*** их ключевое отличие заключается в использовании карты нормалей, в первом случае оная не используется и свет получается плоским как и следует из описания.


Пример с объемным светом.


Пример с плоским светом.

# 6 | Group

Данный пакет предназначен для упрощения работы с игровыми объектами на всех этапах их жизни посредством конфигурационных файлов (в стандартной реализации движка это json). На данный момент этот пакет представлен тремя субпакетами:

- ***map*** - ответственный за игровые объекты
- ***render*** - ответсвенный за процесс отрисовки объектов
- ***container*** - ответственный за делегирование первых двух

## 6.1 Container

Представлен тремя основными интерфейсами, а так же их реализациями по умолчанию посредством которых осуществляется работа. Интерфейсы вместе с имплементриующими классами:

- ***(I) EscapyGroupContainer: (C) DefaultGroupContainer***
- ***(I) EscapyLocationContainer: (C) DefaultLocationContainer***
- ***(I) EscapyRendererContainer: (C) DefaultRendererContainer***

Классы ***DefaultLocationContainer*** и ***DefaultRendererContainer*** имеют конструкторы с модификатором доступа ***protected*** потому их невозможно заинстанцировать на прямую, вместо этого надо использовать класс ***DefaultGroupContainer***.

### 6.1.1 DefaultGroupContainer

Основной класс контейнера объектов реализующий интерфейс ***EscapyGroupContainer*** от которого содержит метод ***boolean initialize();***, который следует самостоятельно и однократно за весь жизненный цикл приложения, вызвать во время инициализации оного.

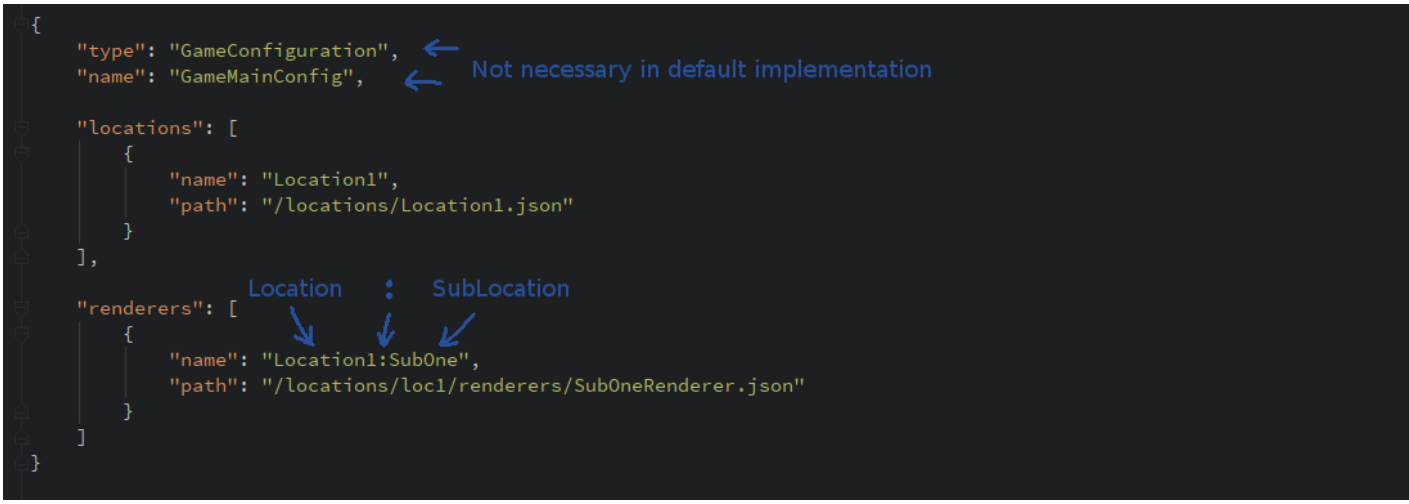
```
@Override
public void show() {
    sprite = new Sprite(new Texture(logoUrl));
    camera.setCameraPosition( x: sprite.getWidth() * .5f,  y: sprite.getHeight() * .5f,  absolute: true);

    new Thread(() -> initialized.set(groupContainer.initialize())).start();
}
```

Пример вызова метода в новом потоке во время стартового экрана приложения.

### 6.1.2 Сериализация

В случае с классом ***DefaultGroupContainer*** для сериализации используется json файл который имеет следующую структуру:




В массиве ***locations*** следует указать имя и путь к файлу из которой должна загружаться локация, в массиве ***renderers*** так же следует указать путь файла из которого будет загружатся ***renderer***, однако имя должно содержать название локации и имя сублокации для ***renderer'a*** разделенное двоеточием.
В конструкторе ***DefaultGroupContainer*** следует указать путь на json файл объекта, а так же загрузчики ***DefaultLocationLoader*** и ***DefaultRendererLoader***

```
public DefaultGroupContainer(String configFile,
                             DefaultLocationLoader locationLoader,
                             DefaultRendererLoader rendererLoader) {
    this.rendererLoader = rendererLoader;
    this.locationLoader = locationLoader;
    this.configFile = configFile;
}
```

Для создания инстанций загрузчиков рекомендуется использовать предназначенные для этого строители: ***DefaultLocationLoaderBuilder*** и ***DefaultRendererLoaderBuilder***.

## 6.2 Map

Этот субпакет отвечает за игровые объекты, он представлен основными интерфейсами:

- ***EscapyLocation*** - интерфейс основной локации
- ***EscapySubLocation*** - интерфейс сублокации внутри основной локации
- ***EscapyLayer*** - интерфейс слоев внутри сублокации
- ***EscapyLayerShift*** - интерфейс смещения слоя
- ***EscapyLayerShiftLogic*** - интерфейс логики смещения слоя
- ***EscapyGameObject*** - интерфейс игровых объектов внутри слоев
- ***EscapyGameObjectRenderer*** - интерфейс рендерера игровых объектов

### 6.2.1 DefaultLocationLoaderBuilder

Каждый из вышеперечисленных интерфейсов имеет свою имплементацию по умолчанию, и что бы собрать их вместе в одну целую и рабочую группу следует воспользоваться строителем ***DefaultLocationLoaderBuilder*** который создаст инстанцию имплементирующую ***DefaultLocationLoader*** которая в свою очередь сможет загрузить локацию из json файла. При создании объекта класса ***DefaultLocationLoader***, тот в конструкторе получает инстанцию очередного загрузчика ***DefaultSubLocationLoader***, а тот в свою очередь инстанцию ***DefaultGameObjectLoader***.

В целом это выглядит следующим образом:

- ***DefaultLocationLoader***
  - ***DefaultSubLocationLoader***
    - ***subLocationLayerShiftLogicAttributeLoader***
    - ***subLocationLayerAttributeLoader***
    - ***subLocationAttributeLoader***
    - ***DefaultGameObjectLoader***
      - ***gameObjectAttributeLoader***
  - ***locationAttributeLoader***

Как можно заметить в конструктор основных загрузчиков так же передаются загрузчики аттрибутов обозначенных в json файле полем-массивом ***"attributes": [...]***, эти загрузчики являются имплементациями ***EscapyInstanceLoader<T>***.

### 6.2.2 Сериализация EscapyLocation

Стандартной имплементацией интерфейса ***EscapyLocation*** является класс под названием ***Location***, который загружается из простого json файла вида:

```
{
    "type": "Location",
    "name": "Location1",

    "attributes": ["", "", ""],

    "subLocations": [
        {
            "name": "SubOne",
            "path": "/loc1/SubLocationOne.json"
        }
    ]
}
```

### 6.2.3 Сериализация EscapySubLocation

Сублокации за которые отвечает интерфейс ***EscapySubLocation*** и его стандартная имплементация ***SubLocation*** загружаются из json файла вида:

```
{
    "type": "SubLocation",
    "name": "SubOne",

    "attributes": [],

    "renderGroups": [],
    "layers": []
}
```

#### RenderGroups

Массив ***RenderGroups*** содержит группу объектов которые будут отрисовываться на графическом конвейере в таком порядке в каком они расположенны в массиве. Каждый объект из такой группы содержит имя, а так же массив слоев.

```
    "renderGroups": [
        {
            "name": "foreground",
            "layers": [
                "layer_foreground_0"
            ]
        }
    ],
```